An engineering process for security patterns application in component based models

Rahma Bouaziz
IRIT, University of Toulouse
Toulouse, France
rahma.bouaziz@irit.fr

Slim Kallel
ReDCAD, University of Sfax,
Sfax, Tunisia
slim.kallel@fsegs.rnu.tn

Bernard Coulette
IRIT, University of Toulouse
Toulouse, France
bernard.coulette@irit.fr

Abstract—Security engineering with patterns is currently a very active area of research. Security patterns − an adaptation of Design Patterns to security – capture experts' experience in order to solve recurrent security problems in a structured and reusable way. In this paper, our objective is to describe an engineering process, called SCRIP (SeCurity patteRn Integration Process), which provides guidelines for integrating security patterns into component-based models. SCRIP defines activities and products to integrate security patterns in the whole development process, from UML component modeling until aspect code generation. The definition of SCRIP has been made using the OMG standard Software and System Process Engineering Meta-model (SPEM). We are developing a CASE tool to support that process.

Keywords— Component; Component based systems; Security patterns; Process; Aspects; SPEM

I. INTRODUCTION

A *software security pattern* [1] is defined as a generic well defined security solution proposed by software security experts to solve a recurrent problem in a given context. Using security patterns, developers can address security issues such as authentication, access control and security. Along with increasing popularity of security engineering with patterns, it is necessary to provide directives and guidelines helping system designers – who are not necessary security experts − apply security patterns. So far, there is no clear, well-documented and accepted process dealing with their full integration from earliest phases of software development until production of application code [2].

Component-based approach is a powerful means to develop and reuse complex systems. In this paper, we take component based software systems as an application domain for our approach. **More precisely, this paper investigates** how security patterns can be integrated into component-based models. Our approach assumes that the security specialists have defined the security patterns and the corresponding solutions. Thus, it provides software designers with capabilities to deploy these solutions. In the context of our application domain, we also suppose that software designers have minimal expertise in security solution domain (authentication, access control, etc.).

Our main goal is to propose a process, called SCRIP (SeCurity patterRn Integration Process) and an associated tool for automatically integrating security patterns into component-based models, and producing an executable secure code. This integration is performed through a set of transformation rules. The result of this integration is a new model supporting security concepts. It is then automatically translated into aspect-oriented code related to security. These aspects are then woven in a modular way within the functional application code to enforce specified security properties. The use of aspect technology in the implementation phase guarantees that the application of security patterns is independent from any particular implementation. In order to provide a clear comprehension of the SCRIP process, we have described it in SPEM [3]. SCRIP is a collaborative process since it is performed by actors − playing different roles – who work together all along the process enactment.

The paper is structured as follows: in the next section we briefly present the SPEM standard and its main concepts. Section III details the proposed process for security pattern integration (SCRIP). Section IV discusses related works. Finally, section V, concludes the paper.

II. SOFTWARE AND SYSTEM PROCESS ENGINEERING METAMODEL (SPEM)

SPEM [3] is the software process modeling OMG standard. SPEM meta-model allows describing software development processes. Its purpose is also to allow processes reuse and documentation. It is structured as both a meta-model conforms to MOF and a UML profile.

Hereafter, SPEM terminology is used to specify the phases, roles and steps that are used to describe the SCRIP process. One of the most important principles of SPEM 2.0 is the distinction between MethodContent elements (mainly TaskDefinition, RoleDefinition and WorkProductDefinition) and Process Space (mainly Activity, TaskUse, RoleUse, WorkProductUse). More precisely, method content elements are generic reusable elements described via the package MethodConent, whereas process elements reuse them via packages ProcessWithMethods and ProcessStructure. For example, several instances of TaskUse may reuse the same instance of TaskDefinition. Concretely, a Process is composed of Activities, which can contain other Activity instances and MethodContentUse elements (Task Uses, Role Uses, WorkProductUses).

In the following, we present the SPEM meta-classes that are used to describe our process. *TaskUse* () describes a piece of work performed by one RoleUse, which may consist of atomic elements called Steps. *RoleUse* () defines responsibilities over specific WorkProducts, which are consumed/produced in specific activities. *WorkProductUse* (generally called artifact ) is anything (piece of information, document, model, source code, etc.) produced, consumed, or modified by a process. We also reuse the concepts of Phase and Lifecycle – inherited from SPEM 1−which are now defined in the SPEM 2.0 plug-in. *A Phase* () is a specialization of WorkDefinition such that its precondition defines the phase entry criteria and its goal defines the phase exit criteria.

We present below an overview of the SCRIP process. Following sections give a detailed description of the three phases of our process. Moreover, to complete the usual graphical notation, we propose a textual notation – conform to SPEM specification −to describe a process structure, as shown below in Fig.1.

```
Process:
    Activity {Kind: Phase}:

        Activity {kind: Iteration}: First
          TaskUse :
            ProcessPerformer {kind: primary}
              RoleUse:
            WorkDefinitionParameter {kind : in}
              WorkProductUse:
            WorkDefinitionParameter {kind : out}
              WorkProductUse:
            Steps
              Step:
              Step:
```

Fig. 1. Textual SPEM 2.0 Process structure description

III. **SCRIP**: **S**e**C**urity Patte**R**n **I**ntegration **P**rocess

The security patterns integration process (SCRIP) is composed of a sequence of phases that cover the whole lifecycle from design to implementation.

A. Overview of SCRIP process

SCRIP is structured as three consecutive phases: Elicitation, Modeling and Implementation. The iterative style should be applied to every phase of our process, but due to space limitation, we describe only one iteration in this paper.

Four process *RoleUse* have been identified: «Security Specialist» who is supposed (in our applicative context) to have some knowledge in component based engineering, «Software designer» who is also required to have minimal expertise in security solution domain, «Transformer» (here a software tool) that automatically applies Security application rules to the application model and finally «Weaver» (here a software tool) that takes application functional code and aspect code as input and delivers a single secure code).

The SCRIP process begins with the Elicitation phase. *WorkProductUse* «Security patterns» and «Component meta-model» are supposed to be available as inputs of this phase. *WorkProductUse* «Security patterns» contains specification and design of specific security solutions. This *WorkProductUse* is used, by applying *TaskUses* T1.1 and T1.2 to obtain two outputs: (1) «Security profile» which extends the component meta-model with new concepts related to security patterns; (2) «*Security Pattern Application Rules (SPARs)*» which is a set of rules expressed in a transformation language like ATL [4].

The Modeling phase includes two *TaskUses*: T2.1 and T2.2. *TaskUse* T2.1 takes as input the *WorkProductUse* «Component meta-model» and produces an «Application Component Model», while T2.2 takes as inputs the «Application Component Model» *WorkProductUse* obtained from *TaskUse* T2.1 and the SPARs *WorkProductUse* obtained from the previous phase to produce a «Secure Application Component Model».

Finally, the implementation phase is carried out through *TaskUses* T3.1, T3.2 and T3.3. *TaskUse* T3.1 takes as input the «Secure Application Component Model» *WorkProductUse* obtained from T2.2 and produces the «Application functional code», while T3.2 generates «Aspect code». T3.3 takes as inputs *WorkProductUse* «Application functional code» and «Aspect code», and produces the «Secure application code ».

To illustrate the enactment of the SCRIP process for a given application, we have chosen a basic GPS case study in which we identified requirements for access control to services offered by components. In this example, we mainly consider the management of access control to various services offered by phone operators; especially downloading geographic maps and managing secure access to satellites.

B. Elicitation phase

As described above, the Elicitation phase comprises two *TaskUses*: *TaskUse* Define Security Profile (T1.1), takes as input the *WorkProductUse* «Security patterns» and «Component meta-model»; *TaskUse* Define Security Pattern application rules (T1.2) takes as input the outputs of *TaskUse* T1.1, i.e., «Security profile» and «Security patterns» *WorkProductsUse*. To carry out the above *TaskUses*, several steps are performed for each *TaskUse*, as we can see in Fig.2.

1) *TaskUse* Define Security Profile

TaskUse T1.1 (Fig.3) is performed by the *RoleUse* «Security and component specialist». Inputs are «Component meta-model» (that defines primitives and basic concepts of model component-based applications) and «Security patterns» (a structured way for collecting security solutions). Several steps are necessary to produce its output *WorkProductUse* that is «Security profile».

The definition of Security profile consists in mapping [5] [6] the concepts of the chosen security patterns with concepts of the component meta-model.

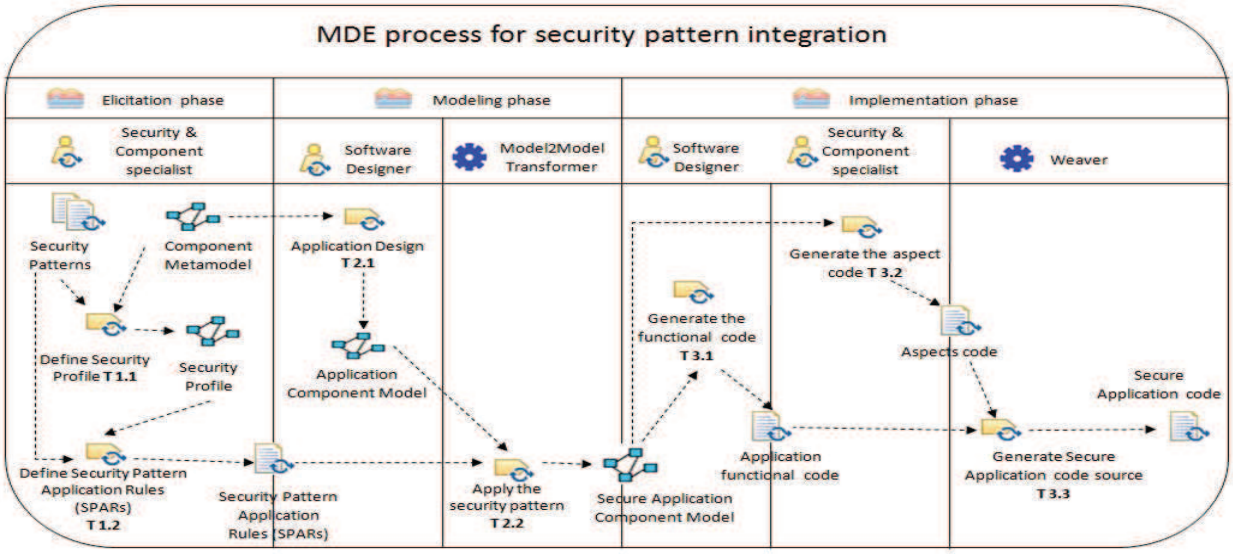

Fig. 2. SPEM2 structural diagram describing the SCRIP process for security pattern integration

As shown in Fig.3, this activity is part of Eliciting Phase of the SCRIP process. As already mentioned, SCRIP is highly iterative but here we only consider a single iteration (First iteration) in order to make more understandable the process. In Fig.3, SPEM meta-model classes, associations and attributes are represented in simple *times* font while the corresponding instances appear in **bold** *times* font.

To address the security issue for the GPS system, we have taken as an example the RBAC pattern (Fig. 4) which provides a solution for access control.

Process: **SCRIP**
Activity {Kind: Phase}: **Eliciting**

Activity {kind: Iteration}: **First iteration**
TaskUse : **Define Security Profile**
ProcessPerformer {kind: primary}
RoleUse:**Security and Component Specialist**
WorkDefinitionParameter {kind : in}
WorkProductUse: **Component Metamodel**
WorkProductUse: **Security Pattern**
WorkDefinitionParameter {kind : out}
WorkProductUse: **Security Profile**
Steps
Step:**Select a security pattern to apply**
Step:**Map pattern participants with component metamodel concepts**
Step:**Identify stereotypes from pattern participants**
Step: **Identify metaclass to extend**
Step: **Add tagged values to stereotypes**
Step: **Associate stereotypes with adequate UML metaclasses**

Fig.3. Define Security profile TaskUse (T1.1)

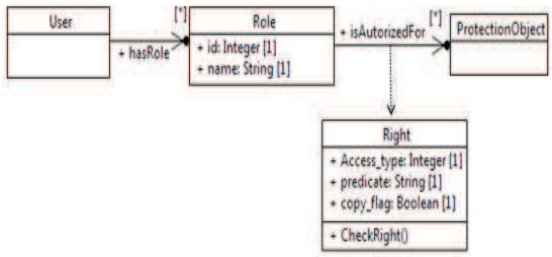

Fig. 4. RBAC pattern (adopted from [1])

As an example, we present in Fig.5. a subset of the «Security UML profile» produced for Role-based Access Control policy pattern (RBAC). It is mainly a set of stereotypes derived from some meta-classes of the component meta-model.

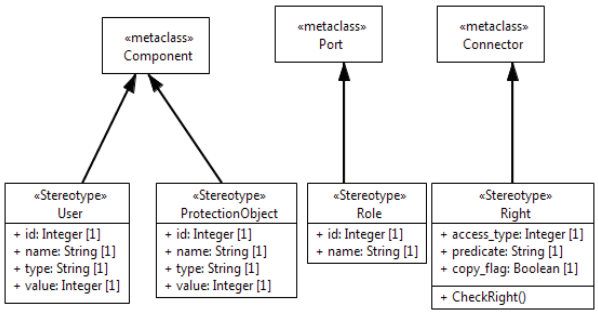

Fig. 5. UML profile for RBAC pattern

2) *TaskUse* Define Security Pattern Application Rules

Fig.6 depicts the Define Security Pattern application rules *TaskUse,* which is performed by the *Security and component specialist* RoleUse.

```
Process: SCRIP
    Activity {Kind: Phase}:  Elicitation

      Activity {kind: Iteration}: First iteration
        TaskUse :  Define Security Patterns Application Rules
          ProcessPerformer {kind: primary}
            RoleUse:Security and Component Specialist
          WorkDefinitionParameter {kind : in}
            WorkProductUse: Security Profile
WorkProductUse: Security Patterns
          WorkDefinitionParameter {kind : out}
            WorkProductUse: Security Patterns Application Rules
          Steps
            Step:Define parametre(s) that software developer should
                 input
Step:Map  a set of concepts among security participants
 Step:Identify stereotypes from pattern participants
Step: Look form common roles between the to-be-applied
                 pattern and the application concepts
```

Fig. 6. Define Security Pattern Application rules TaskUse (T 1.2)

We have defined a set of rules (SPARs for Security Pattern Application Rules) to automate the integration of security patterns into software components. These rules are deduced through the relationships between security concepts of the selected patterns and the corresponding UML profile. These rules are applied in two steps: (1) Ensure the correspondence between the main pattern concepts and the correspondence model elements (specified as a component, a connection or a port). For each security pattern, we select the main concept that should be applied by the designer (i.e., the name of the application artifact that corresponds to the role of the applied security pattern). The definition of this correspondence depends on security patterns previouslyapplied to the same model; (2) The second step corresponds to the mapping of other security pattern concepts to the corresponding model elements. Concretely, this mapping is performed by applying the respective stereotypes defined in the corresponding UML profile. We have implemented this mapping as a model-to-model transformation using ATL.

Fig.7 shows an excerpt of the ATL code corresponding to RBAC pattern application rules.

C. *Modeling phase*

This phase produces a secure component model after having applied one or several security patterns to an initial application component model. This modeling phase is carried out by means of two *TaskUse*: Application Design (T 2.1) and Apply security pattern (T 2.2).

1) *TaskUse* Application Design

The goal of T2.1is to model the functional application design. The «Software Designer» *RoleUse* carries out this *TaskUse*. The designer may use the Papyrus suite tool[7], for example, to specify his application using UML2 component diagram. He may also use any UML profile that supports specific component models like CCM, EJB or Fractal. The resulting component model does not support any security concept.

```
helperdef : getStereotype(p : UML2!Profile, name : String) :
 UML2!Stereotype =
p.ownedStereotype->select(s | s.name=name)->first();
rule Package {....
do {
t.applyProfile(UML2!Package.allInstances()->select(s |
s.name='RBACProfile')->first());
thisModule.entityProfile<-UML2!Package.allInstances()->
select(s | s.name=' RBACProfile ')->first();
}
rule Component {....
do{
if(s.name='SecureSattelite'){
t.applyStereotype(thisModule.getStereotype(thisModule.
entityProfile,'ProtectionObject));
      }
}
rule Port{
do{
if(s.clientDependency.isEmpty() ands.owner.name='GPSTerminal'){
:t.applyStereotype(thisModule.getStereotype(thisModule.entityProfile,'Role))
}
}
```

Fig 7. Part of RBAC pattern application rules

Fig. 8 provides a case in point of a component-based application using a UML component diagram for our GPS case study. It contains five components:

-*Satellite* enables to emit permanently a navigation message containing all the necessary data for the receiver to perform navigation calculations.

- *SecureSatellite* emits secure signals.

- *GPSTerminal* receives the message transmitted by the Satellite and must have access rights to the signals sent by SecureSatellite. It requires the map downloading service from the Operator component.

- *Operator (Phone operator)* offers the service to download maps. It requires maps requested by members from MapDataBase.

-*MapDataBase* offers to the Operator the possibility to have the map downloaded by the applicant.

To address the security issue for the GPS system, we have taken as an example the RBAC pattern which provides a solution for access control.

1) *TaskUse* Apply the security pattern

TaskUse T2.2 is performed to obtain a secure application component Model (Fig. 9). This *TaskUse* takes as input two *WorkProductsUse*: «Application Component Model» and «Security Pattern Application Rules». The «Model2Model transformer» *RoleUse* carries out this *TaskUse*.

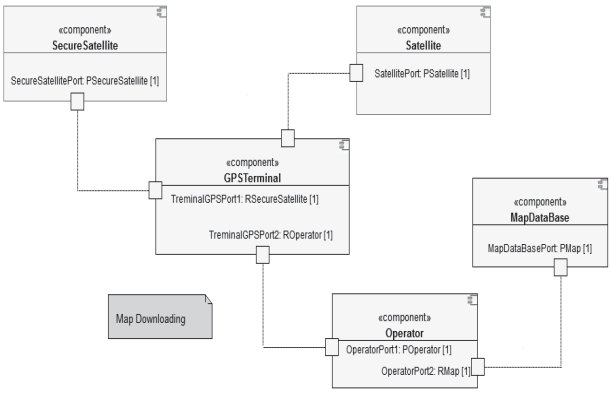

Fig. 8.Component Diagram of Basic GPS system

Process: **SCRIP**
Activity {Kind: Phase}: **Modeling**
Activity {kind: Iteration}: **First iteration**
TaskUse : **Apply the security Pattern**
ProcessPerformer {kind: primary}
RoleUse:**Transformer**
WorkDefinitionParameter {kind : in}
WorkProductUse: **Application Component Model**
WorkProductUse: **Security Patterns Application Rules**
WorkDefinitionParameter {kind : out}
WorkProductUse: **SecureApplication Component Model**
Steps
Step:**Ensure the correspondence between the main pattern concepts and the corresponding model elements (specified as a component, a connector or a port)**
Step:**Select the main concepts that should be applied by the designer**
Step: **Map the other security pattern concepts to the corresponding model elements**

Fig. 9. Apply the security pattern TaskUse (T2.2)

Once steps of *Activity* 2.2 are executed, we get a secure component model as output *WorkProduct (*Fig. 10).

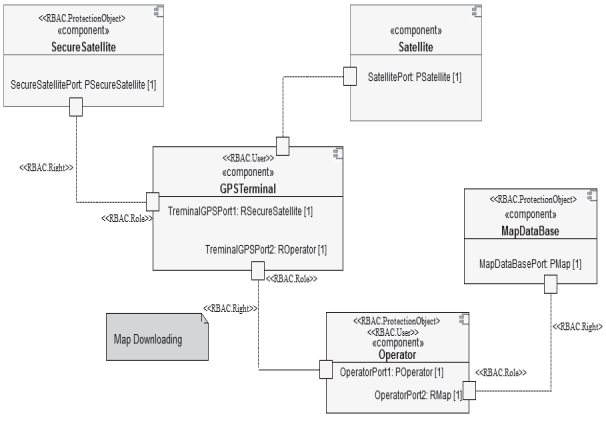

Fig. 10. Secure Component Diagram of basic GPS system

D. Implementation phase

This phase is dedicated to produce a secure application code through three *TaskUse* T3.1, T3.2 and T 3.3. This implementation stage includes generating two intermediate artifacts: the «Functional code» of the component based application and the «Aspect code».«Security and component specialist» and «Software designer» cooperate to define the final secure application code as explained below in the tasks.

1) *TaskUse* Generate the functional code

TaskUse 3.1 aims to generate the «Functional code» of the component based application. We reuse existing approaches for functional code generation. Indeed several approaches and commercial tools support the generation of code skeleton with different technologies (EJB, .NET, C++, etc.) from a UML component diagram, based on a set of predefined libraries. The designer can also produce the corresponding code by using for instance the MDA approach. It first transforms the application component model into a platform specific model. The corresponding code is then generated using a model-to-text generator.

2) *TaskUse* Generate the aspect code

TaskUse 3.2 takes as input the secure application model to define aspects. During this *TaskUse,* the «Security specialist» and the «Software designer» *RoleUses* collaborate to generate «aspect code». For each security pattern, we propose a template to generate AspectJ code and a helper as a set of Java classes. We generate a skeleton of aspect code, which should be completed by the developer, according to the functional application code generated during the *TaskUse* 3.1.*Pointcuts* intercept the call of critical methods, while *advices* ensure the functionalities of the patterns. We generate different types of advices (around, before, and after) depending on the security pattern.

For example, the generated aspect code for RBAC policy is defined as following. The generated pointcut intercepts the call of all methods performed by the system users. We generate an around advice, which verifies before the execution of each intercept method, that the caller (user) has the expected role (i.e. has the access) to execute this method.

3) *TaskUse* Generate secure application code

TaskUse T3.3 takes as input the «application functional code» and the «aspects code» to produce a «code for secure application». Secure application code is obtained by weaving aspect code resulting from the 3.2 *TaskUse* with the application functional code obtained from the 3.1 *TaskUse*.

IV. RELATED WORKS

There are a large amount of works addressing the topic of security design patterns applicability and usability.

Ortiz et al. [8] provide an analysis of the main works related to security patterns. They discuss their applicability for the analysis and design of secure architectures in real and complex environments. Here, we sum up some of the proposals for integration of security patterns. In [9], authors propose a security pattern integration technique dealing with model transformation using ATL. Moreover, authors in [10] use Petri nets to model security patterns at an abstract level.

A methodology for integrating security patterns into all stages of the software development lifecycle is proposed in [11]. Other approaches [12] [13] present the use of aspect oriented software design approach to model security patterns as aspects and weave them into the functional model.

Concerning design pattern instantiation, S. Yau [14] uses a formal design pattern representation and a design pattern instantiation technique for automatic generation of component wrappers from design patterns. In addition, several approaches introduce their own tool-based support for pattern instantiation. In [15] authors propose an approach for representing and applying design patterns. In [16] authors provide an UML profile which allows the explicit representation of design patterns in UML models through a model transformation approach. Authors in [17] describe an approach for creating automated transformations that can apply a design pattern to an existing program. In [18], authors propose a method supporting design patterns application in software projects, based on a semantics defined via UML profile and model transformations.

We can conclude that most of existing approaches focus on the application of security patterns at design level without providing any mechanism for implementing them in component-based applications. There is little work concerning the full integration of security patterns from the earliest phases of software development and providing automatic generation of the final secure application code. Even more, the code that applies security patterns is generally not well modularized, as it is tangled with the code implementing each component's core functionality and scattered across the implementation of different components.

To remedy these limitations we have provided a security pattern integration process –described in SPEM −with tool support in order to encourage developers to take advantage from security solutions proposed in security patterns.

V. Conclusion and Future work

Security is one of the most important properties to consider in complex systems development. A promising way to address this issue is the application of solutions proposed by security patterns. A large set of security patterns have been defined by security experts. In this paper, we have addressed the problems related to the applicability and usability of security patterns in component-based applications. In this context, we have proposed a structured process, called SCRIP, to build secure component based systems using patterns. To apply such patterns and produce executable secure code in an iterative way, we use aspect-oriented techniques.

As future work, we aim to provide a complete development environment to design secure component based application using the proposed SCRIP engineering process.

Acknowledgment

We would like to thank Komlan Akpédjé Kedji for his valuable comments and suggestions.

References

[1] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad, Security Patterns: Integrating Security and Systems Engineering. Wiley, 2006, p. 600.

[2] P. T. Devanbu and S. Stubblebine, "Software Engineering for Security : a Roadmap," in Proceedings of the conference of The future of Software engineering, 2000.

[3] "SPEM 2.0." http://www.omg.org/spec/SPEM/2.0/.

[4] "ATL." http://www.eclipse. org/atl/.

[5] R. Bouaziz and B. Coulette, "Applying Security Patterns for Component Based Applications Using UML profile," in proceedings of the *15th IEEE International Conferences on Computational Science and Engineering*, 2012, pp. 186–193.

[6] R. Bouaziz and B. Coulette, "Secure Component Based Applications Through Security Patterns," in proceedings of the *Workshop on Security of Systems and Software Resiliency*, 2012, pp. 749–754.

[7] "Papyrus UML." http://www.papyrusuml.org

[8] R. Ortiz, S. Moral-García, S. Moral-Rubio, B. Vela, J. Garzás, and E. Fernández-Medina, "Applicability of security patterns," in *On the Move to Meaningful Internet Systems: OTM 2010*, 2010, pp. 672–684.

[9] Y. Yu, H. Kaiya, H. Washizaki, Y. Xiong, Z. Hu, and N. Yoshioka, "Enforcing a security pattern in stakeholder goal models," in *Proceedings of the 4th ACM workshop on Quality of protection*, 2008, pp. 9–14.

[10] V. Horvath and T. Dörges, "From security patterns to implementation using petri nets," in *Proceedings of the 4th international workshop on Software engineering for secure systems*, 2008, pp. 17–24.

[11] E. B. Fernandez, M. M. Larrondo-Petrie, T. Sorgente, and M. Vanhilst, "A Methodology to Develop Secure Systems Using Patterns," *Integrating Security and Software Engineering*, 2006, vol. 5, pp. 107–126.

[12] R. France and I. Ray, "Using Aspects to Design a Secure System," *Proc. of the 8th IEEE International Conference on Engineering of Complex Computer Systems*, 2002, pp. 117– 126.

[13] R. F. Indrakshi Ray, "An aspect-based approach to modeling access control concerns," *Information and Software Technology*, vol. 46, pp. 575–587, 2004.

[14] S. S. Yau, "Integration in component-based software development using design patterns," in *Proceedings of the 24th Annual International Computer Software and Applications Conference.* 2000, pp. 369–374.

[15] G. El Boussaidi and H. Mili, "A model-driven framework for representing and applying design patterns," in Proceedings of the *31st Annual International Computer Software and Applications Conference - Vol. 1-*, 2007, pp. 97–100.

[16] X.-B. Wang, Q.-Y. Wu, H.-M. Wang, and D.-X. Shi, "Research and Implementation of Design Pattern-Oriented Model Transformation," in *Proceedings of the International Multi-Conference on Computing in the Global Information Technology*, 2007.

[17] M. Ó Cinnéide and P. Nixon, "Automated software evolution towards design patterns," in *Proceedings of the 4th international workshop on Principles of software evolution - IWPSE '01*, 2002, p. 162.

[18] P. Kajsa and L. Majtás, "Design patterns instantiation based on semantics and model transformations," in Proceedings of the *36th Conference on Current Trends in Theory and Practice of Computer Science*, 2010, pp. 540–551.